SOFTWARE FOR A SMALL ON LINE COMPUTER

A Thesis Submitted
in partial fulfilment of the requirements
for the Degree of
MASTER OF TECHNOLOGY

BY

Rajendra Kumar Kanodia

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

August 1968

This is to certify that this work on "Software Development for a Small on Line Computer System" has been carried out under my supervision and it has not been submitted elsewhere for a degree.

(Dr.) H.N. Mahabala
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology
Kanpur

August 1968

ACKNOWLEDGEMENT

The author is gratful to Dr. H.N.Mahabala for his assistance and guidance during this project.

ABSTRACT

This report describes the development of an assembly programming system for the Trombay Digital Computer TDC-12. The essential features of the TDC-12 Assembly Language TAL are symbolic and self relative addressing system, data declaration psuedo-operations for decimal and octal data items, text facility, and storage assignment psuedo-operations. The TAL assembler makes two passes through a source program written in language TAL and translates it into machine language of the TDC-12. The TAL assembler has itself been written in the language TAL. Machine language version of the TAL assembler was produced by another assembler written in IBM 7044 MAP language. A scheme which will permit inclusion of user defined Macros, in the present framework of the assembler, is given.

CONTENTS

CONTENTS(Contd.)

CHAPTER ONE

INTRODUCTION

The last few years have seen a phenomenal and continue advance in the computer technology. Improved and more efficient computers, which are much better equipped to handle problems of ever increasing complexity, are replacing the old computers. A problem created by these rapid advances in hardware design is that computers tend to become obsolete in a very short time. To get the maximum use out of the computer it is necessary therefore that the computer be put to use immediately after the hardware is ready. Proper utilization of a computer requires extensive software facilities. Since software development takes considerable time, some software must be developed in advance. The best thing to do is to start writing software as soon as the computer specifications have been laid down.

The extent to which the software should be developed in advance, depends on the particular machine and the type of application for which it is intended. For instance a large scale general purpose computer system operating in a university would require a comprehensive set of compilers compatible with other computer systems. On the other hand a computer used for process control in a petro-chemical complex will normally have a set of programmes designed specifically for the installation.

The Trombay Digital Computer (called TDC-12) is intended to be a small scale computer system with real time application capabilities. The random access core memory holds 4096 12-bit words with a cycle time of 1.5 micro-seconds. The system provides for program interruptions as functions of input output device conditions on a priority basis. Data transfer

between input-output devices and core memory can be initiated by means of data interrupt facility. Upto twelve devices can be connected to an input-output bus. However only two teletype units will initially be linked. The device selection and data transfer for the devices attached to this bus is completely under program control. The instruction set consists of twelve storage reference instructions and three non-storage reference instructions which can be microprogrammed to produce a large variety of instructions.

As the computer is intended for real time data acquisition and process control applications, and because it has a small memory capacity, sophisticated compilers and general purpose languages are not really necessary. A simple algebraic compiler permitting easy programming of data acquisition algorithms in a real time environment and an assembly programming system would be adequate. Process control systems are normally specific to the installation and are meant for long term applications, so they must utilise the machine efficiently. Most of these software systems are therefore written in the assembly language.

This report traces the development of an assembly programming system for the TDC-12.

CHAPTER TWO

DEVELOPMENT OF SOFTWARE FOR A NEW COMPUTER SYSTEM

2.1 Minimum Software: The Assembler

The software for a modern computer system consists of translators, interpreters, supervisory systems, input output routines, a library of general purpose subroutines and many other things. Most of these systems are very much machine dependent and since they are used so often, they must make efficient use of the machine. To write these systems, therefore, there is a need for a language which will give the ability to express any machine language algorithm and yet be simple enough to relieve the programmer of the tedious task of machine language programming. Assembly languages are, in general, meant to serve precisely this purpose.

A statement in an assembly language program is essentially a machine instruction expressed symbolically. The numeric operation code is replaced by a suitable nemonic which is easy to remember. The address portion of the instruction is also expressed symbolically. The symbol in the address portion is associated with the instruction or data to which it is supposed to refer. The numeric equivalent of this symbol is the location in which this instruction or data is to be placed. To further simplify the job of the programmer, data declaration facilities may also be included in the assembly language. The assembly program then assigns storage locations, translates operation codes, evaluates addresses, 'assembles' the operation code and address, interpretes data, and produces a machine version of the symbolic program.

Assembly language programming systems form the core of even the

most sophisticated computer software systems. The first step in the development of software for a new computer is, therefore, to write an assembly language programming system. In what language should the assembly system itself be written? The same considerations that are applicable to other software systems, apply to the assembly system also. The assembly system should, therefore, be written in assembly language or in other words, its own language. This has other advantages too. The writing of an assembly system in its own language provides valuable feeback for its own design. Once an initial version is operating, improvements can be made by a basic bootstrap technique. The next section in this chapter discusses various method for obtaining a machine version of an assembler for a computer which is not yet operation.

2.2 Methods of Writing an Assembler for a New Computer

The task of developing an assembler for a new computer can be handled in two ways. The first uses a bootstrap technique and the other makes use of the software facilities offered by existing computers.

2.2.1 First Method

In this method that uses a bootstrap technique, a series of assemblers, each obtained by modifying the previous version to accept source programs in an upgraded assembly language, are written. The various steps are explained below:

1. Choose a minimum assembly language, which though minimal in the sense of complexity is sufficient to describe the symbolic version of any machine language program.

2. Write an assembler in the minimum assembly language which will

translate any source program written in the minimum assembly language into the machine language.

3. Hand translate this minimum assembler into the machine language.

4. Modify the assembler written in minimum assembly language to accept a source program written in a second level of assembly language. Note that the assembler itself remains in first level language only.

5. Translate the new assembler using the earlier executable assembler to obtain an executable version of the modified assembler.

Before applying step four we had a source language as well as machine language version of the minimum assembler. After completing the step five we get a source language as well as machine language version of an upgraded assembler to accept source programs written in a higher level assembly language. Successive applications of steps four and five can, therefore, be used to obtain a source language and machine language pair of assemblers which would translate source programs written in an assembly language of any desired complexity.

2.2.2 Second Method

This method, which makes use of the facilities offered by existing computers, can be stated as follows:

1. Write the assembler in the assembly language of the new computer. This assembler accepts source programs written in assembly language and translates them into machine language of the new computer.

2. Write another assembler in a language which can be executed on some available computer. Input-output language pair for this assembler

is again the assembly language and machine language of the new computer.

3. Execute the assembler written in step two with assembler of step one as input. Output will be the machine version of the assembler for the new computer. This assembler can be directly executed on the new computer.

2.2.3 Comparison between the Two Methods

The bootstrap technique has two major disadvantages. Since assembly languages are very low level languages, in general, there may not be any appreciable difference in a minimal assembly language and its final version. This would require hand translation of an entire assembler. The task of machine language coding is very tedious. The other disadvantage comes from the fact that execution of minimal and intermediate assemblers requires existence of the computer. If software should be completed before the computer is ready, a simulator on an existing computer is necessary. Since software for the TDC-12 should be ready before thehardware becomes operational, a simulator on an existing computer is necessary to adopt the bootstrap technique Simulators though useful for debugging purposes, are not economical for production runs because of their inherent inefficient nature.

In the second method, utilization of powerful facilities offered by existing computers simplifies the work a great deal. No manual translation into machine language is required and the machine language version can be produced without the existence of the new computer or a simulator. The task of writing the assembler version to operate on an existing computer is simplified if it has extensive software. Good input-output facilities can make the process of debugging much easier.

To develop an assembler for TDC-12 it was decided to make use of the powerful facilities offered by IBM 7044. An assembler (called TAL44) which would accept TDC-12 Assembly Language (called TAL) programs as its input and produce TDC-12 Machine language version of these programs was written in IBM 7044 MAP language. The input to TAL44 was an assembly program, written in TAL, which would accept a source program in TAL and produce a corresponding TDC-12 machine language program. The output from TAL44 was a machine language translation of the TAL assembler. Since the assembler had to be debugged and also since there was an expectation that hardware design could be based on the experience of the software man, a simulator for TDC-12 was written on IBM 7044. The assembler has been thoroughly debugged.

CHAPTER THREE

TDC-12 ASSEMBLY LANGUAGE TAL

3.1 Introduction

Coding in machine language is a tedious and time consuming job. One of the efforts in simplifying the job is the assembly language. For the most part, an assembly language is similar in structure to machine language, but with mnemonic symbol substituted for each of the binary instruction codes and programmer symbols for the other fields of an instruction. It may also have various added features. Among these features might be a set of psuedo-operations to expand the programming facilities of machine language. Thus, the programmer has available through an assembly program language all of the flexibility and versatility of machine language, plus facilities that greatly reduce machine language programming efforts.

3.2 TAL Features

The use of symbolic names in TAL make a program independent of actual machine locations. Programs and routines written in TAL can be relocated and combined as desired. Routines within a program can be written independently with no loss of efficiency in the final program. Symbolic instructions may be added or deleted without reassigning the storage addresses.

3.2.1 Operations

The TAL provides the user with all of the TDC-12 machine operations, as well as an extensive set of psuedo operations. In TAL machine operation codes are expressed in their mnemonic form. A psuedo operation is any operation included in the TAL that is not an actual TDC-12 machine operation. There are two location counter, four data generation, one storage allocation,

one symbol definition, and two other miscellaneous psuedo-operations in TAL.

3.2.2 Location Counter

During assembly a location counter is used to determine the next location to be assigned to an instruction. For each machine instruction processed by the assembler, the value of the location counter at that time is increased by one. Certain psuedo-operations may result in no increase at all, whereas others may result in an increase of more than one. The location counter is initially set to zero. Using location counter psuedo-operations a programmer may reset the location counter according to his needs.

The location counter psuedo-operations enable the programmer to put together various sections of the program that might have been written separately, in the order he wants.

3.3 Structure of the TAL

A programme in TAL consists of a series of symbolic instructions, last of which must be the psuedo-operation END.

3.3.1 Symbolic Instructions in TAL

A symbolic instruction consists of the following three parts:

1. The Name Field, which may contain a name by which other instructions can refer to the instruction named. In mnemonic instructions, use of this field is optional, and it may be left blank. Specific name field requirements for each of the psuedo-instructions are given later.

2. The Operation Field, which contains the mnemonic machine operation code or psuedo-operation code. A blank operation field results in an error. An asterisk (*) may appear in the operation field immediately

to the right of the last character of a machine operation code. The presence of this character indicates that the operation is indirectly addressed, and the assembler inserts the appropriate bit into the word. Indirect addressing is permitted only with storage reference machine instructions.

3. The Variable Field, which contains the necessary address portion of the storage reference machine instruction, or is left blank in the case of non-storage reference machine instructions. For a psuedo-instruction it contains whatever is specified for that psuedo-instruction.

4. The Comments Field, which is used for explanatory remarks; it exists solely for the convenience of the programmer and does not effect execution of the program.

3.3.2 Symbolic Instruction Format

Symbolic instructions are typed one per line. A line feed marks the beginning of a new line and carriage return end of the current line. Following format applies for various fields.

The name field is five characters long. If name field contains a symbol it must be left justified.

Sixth character is always blank, except in the case of a remarks line with an asterisk (*) in first column.

The operation field begins at seventh character and is five characters long.

Twelfth character is always blank except in the remarks line with an asterisk (*) in the first column.

Variable field beings at thirteenth character and is terminated by the first blank except in the case of psuedo-instruction BCI.

The comments field follows the variable field. It must be preceded by at least one blank to separate it from the variable field. In the absence of variable field the comments field may not begin before the fourteenth character.

3.3.3 Remarks

An asterisk (*) as first character in the line and any desired information in the rest of the line constitute a remark. Remarks are completely ignored by the assembler.

3.3.4 Symbols

A symbol is a sequence of from one to five characters, first of which is an upper-case letter of the English alphabet. Each of the other characters, if any, may be an upper-case letter of English alphabet or a digit.

Symbols are defined by their appearance in the name field of an instruction and the value assigned is the address of the instruction in which they appear. Symbols may be assigned values by the programmer using the symbol definition psuedo-operation EQU.

3.3.5 Decimal Integer

A decimal integer is a string of digits, from 0 through 9, which may be preceded by a plus or minus sign. The decimal integer in TAL must be in the range -2047, 2047.

3.3.6 Octal Integer

Same as Octal except that digits can only be from 0 through 7.

3.3.7 Address

Once an instruction has been named by the presence of a symbol in the name field, it is possible for other instructions to refer to that instruction using that symbol.

An address in TAL may take one of the following forms:

1. Signed or unsigned decimal integer.
2. A symbol which may be followed by a signed decimal integer constant.
3. An asterisk (*) which may be followed by a signed decimal integer constant.

The asterisk (*) in address portion stands for the location of the instruction in which it appears. Thus, it will have different values in different instructions.

3.4 Machine Instructions

Following are the storage reference machine instructions, which can be indirectly addressed and which have an address portion:

AND XOR LAC SAC ADD SUB RAD ISZ JMP JMS CAS XCT.

Following are the non-storage reference machine instructions:

NOP CLA CLC CAR CAL CTR CTL CMC CMA IAC CIA STA ORS SKP
SNC SZC SZA SNA SMA SPA STP IOF ION SMK SMON KSF KCC KRS
KRB KSP KCS KRC KRP TSF TCF TPC TLS TSP TCP TPS TLC.

The non-storage reference machine instructions can neither be indirectly addressed nor can they have address portion.

3.4.1 Rules for Assembly of Machine Instructions

The assembly of machine operations involves the

following functions:

1. If there is a symbol in the name field, this symbol is given the value of the next location to be assigned by the assembler when the instruction is encountered.
2. The operation code is translated into a 12 bit binary instruction word.
3. If direct addressing is specified and allowed the appropriate bit is inserted.
4. If an address is present in variable field, it is evaluated. If the address refers to memory sector zero it is left as such. If it does not refer to sector zero a one is inserted in the sector bit of machine instruction and a comparison is made between the memory sectory in which the instruction is located and the memory sector to which the address refers. If they are not same an error message is typed out. The rightmost six bits of 12 bit address form the relative address in the sector.
5. The instruction with indirect addressing and sector bits already set in combined with relative address in the sector by a logical OR operation.
6. The 12 bit instruction that results is assigned to next location to be assigned by the assembler.

3.5 TAL Psuedo-Operations

A psuedo-operation might be defined as any operation available in TAL that is not an actual machine operation. Psuedo-operations are used to provide information to the assembler for assignment of core storage to various program and data sections by the programmer for data definitions

for specifying end of the program, and for remarks.

3.5.1 Data Generation Psuedo-Operations

The data generation psuedo-operations are used to enter words of data into the program during the assembly. The data might be in the form of decimal constant, octal constant, characters, or an address.

These psuedo-opcodes may be entered any where in the program. Areas for these psuedo-opcodes are automatically assigned core storage locations in the order in which they appear.

OCT: The psuedo-operation OCT is used to enter binary data expressed in the octal form into a program. Variable field contains an octal integer, which is converted to a binary word. This word is assigned a location analogous to machine instructions. If there is a symbol in the name field it is assigned to the word in which data is stored.

DEC: Same as OCT except that there is a decimal constant in the variable field.

DSA: The psuedo-operation DSA is used to enter an address as one word into the program. Variable field contains an address which is evaluated as a 12 bit binary word. The word is assigned location analogous to machine instructions. If there is a symbol in the name field it is assigned to the word in which the address is stored.

BCI: The psuedo-opcode BCI is used to generate a contiguous string of data words containing one character binary representation per word. First item in the variable field is a decimal constant specifying number of words to be generated. The decimal constant is followed by a blank

which in turn is followed by the string of characters to be assembled. A symbol in the location field is optional, if present, it is assigned the location of first word.

3.5.2 Storage Allocation Psuedo-Operations

BSS: This psuedo-operation is used to reserve core storage areas as data storage areas or work areas. BSS performs two functions:

1. A block of consecutive storage locations is reserved. The number of locations reserved is the value of decimal constant in the variable field; the location of the block is determined by the value of the current location counter when the BSS is encountered.
2. If there is a symbol in the name field, it is assigned to the first location of the block reserved.

3.5.3 Location Counter Psuedo-Operation

Location counter psuedo-operations are used to redefine the value of the location counter.

ORG: The effect of this instruction is to reset the location counter to the value of the address in the variable field. The next instruction to be assembled will then be assigned to the new origin. If there is a symbol in the address it must have been defined earlier. A symbol in the name field is optional; if present, will be assigned the value of the new origin.

OGS: If location counter is not at the first location of a memory sector (in which case OGS has no effect) the location counter is reset to the first location of the next memory sector. If there is a symbol in the name field it is defined as the new origin. Variable field is absent.

3.5.4 Symbol Definition Psuedo-Operation

EQU: The EQU psuedo-operation is used to assign to a symbol a value other than that of the value of the location counter in control when the symbol is encountered. The value assigned to the symbol is the value of the address in the variable field. If address contains a symbol it must have been defined earlier.

3.5.5 Miscellaneous Psuedo-Operations

REM: Treats the line as remark and is ignored by the assembler.

END: The END psuedo-operation is used to signal the end of the symbolic deck. The effect of this operation is to terminate the assembly. The variable field may contain an address, which defines the location where program execution starts. If the variable field is left blank, zero is assumed.

3.6 Error Messages

The source program is checked for following errors:

Error 1: Symbol multiply defined.

Aeeor 2: Illegal constant

Error 3: Symbol undefined

Error 4: Address in machine instruction does not refer to the sector zero or the sector in which instruction is located.

Error 5: Symbol table full

An illegal character in the symbol will be detected as Error 2.

CHAPTER FOUR

THE TAL ASSEMBLY PROGRAM

4.1 How does an assembler work?

The specifications laid down for the language TAL require that the TAL assembler be capable of performing the following major functions:

1. Read and identify constants and symbols.
2. Supply actual operation codes for the symbolic machine instructions.
3. Allocate storage to the object program
4. Convert numbers from decimal and octal systems to binary
5. Build and search symbol table
6. Compute values of the address portions involving symbols and constants
7. Assemble related partial words into full words for output.

The symbolic instructions are sequentially supplied to the assembler. Each machine instruction and data word can be assigned an object time storage location during this sequential processing. Storage allocation in the case of certain psuedo-instructions is, however, a function of the value of the symbols appearing in the variable fields of these instructions. In order that storage allocation may be completed in one sequential examination of the source program, the limitation has been laid in the language TAL that all such symbols must have been defined previously. Note however, an attempt to evaluate the address portion of the instruction simultaneous with storage location may not be successful. This is because address portion may contain symbols which have not been defined as yet.

The assembly requires scanning of the source deck one or more times. The assembler is referred to as an 'n' pass assembler if the number of scans required is n. Following is a discussion of a one pass and a two pas assemblers.

One pass assembler: Assemble and write in object deck those instructions which either do not have an address portion or whose address portion can be evaluated simultaneous with storage allocation. For those instructions which can not be assembled at this time, information about their object time storage location, operation code, symbolic reference, and the constant in address portion is saved in a table. These instructions can be assembled after the whole source program has been examined once and storage allocation completed. This is a one pass solution.

Two pass assembler: One sequential examination of the source program allocates storage for each instruction and data; and defines all the symbols in the name fields. During a second pass through the source program, the machine instructions are assembled and data generated.

Several pass assemblers: Big sophisticated computer systems have several pass assemblers and delegate a major portion of the assembly to the loader. Thse assemblers provide facilities such as relocatable object programs, externally defined symbols, standard I/O macros, macros for subroutine linkages etc.

The one pass solution, though efficient in certain sense requires a large memory capacity on two accounts. One, for table of unassembled instructions and two, for the complicated programs to deal with a complex

pass solution did not seem feasible and a two pass solution was, therefore, the only choice for the TAL assembler.

4.2 Pass One, Storage Allocation

Purpose of the first pass is to assign storage to each machine instruction and data word, and thereby define all symbols appearing in the name fields. A two state switch called PASS is set to indicate that control is in the hands of routine PASS1. Initially, the location counter is set to the first location that will be available to the object program at execution time. Then a symbolic statement is read. Remark cards are ignored. The symbolic operation code in the statement is compared against those in the psuedo-operation table. If it is a psuedo-operation control is transferred to the appropriate routine that will analyse this psuedo-operation. If the symbolic operation is not a psuedo-operation it is assumed to be a machine instruction. If there is any symbol in the name field, a search is made through the symbols defined earlier. In case the symbol has already been defined an error message is typed out. If no error, the symbol along with the value of the location counter is entered in the symbol table. The location counter is incremented by one and another card read. The process continues until an END psuedo-operation is encountered. Pass one is terminated and control returns to the calling program. Each psuedo-operation requires a special action to be taken during pass one. OCT, DEC, and DSA are processed similar to the machine instructions. In the case of BCI and BSS psuedo-operations during pass one, the only concern is the number of words of storage required and the definition of any symbol appearing in the name field. First, the symbol is defined and then the variable field is processed to determine the number of words to be reserved. The location counter is

incremented by this amount. In case of ORG, the variable field is evaluated and the location counter is reset to its new value. If there is a symbol in the name field, it is assigned the new value of the location counter. OBS is processed similar to ORG, except that there is no variable field in this case and the new value of the location counter is a function of the previous value. EQU does not effect the location counter. The symbol in the name field is assigned the value of the address in the variable field. Finally the END psuedo-operation terminates pass one and returns control to the calling program. (See the flow charts)

4.3 Pass Two, Assembly

When all the symbols have been defined by pass one, it is possible to finish the assembly by processing each symbolic instruction in order, and determining values for its operation code and for its variable field. The purpose of pass two is to evaluate the operation code and address portion of each machine instruction, to assemble the binary machine word, and to output this binary word in the object deck. For data generation psuedo-operations, the data must first be converted into its binary equivalent and then output into the object deck. The basic procedure is similar to pass one although pass two is much more complicated than pass one.

To begin with switch PASS is set to indicate the second pass. Location counter is again set to the first location that will be available to the object program during execution time. A symbolic statement is now read. If the statement refers to a psuedo-operation, an appropriate routine that will analyse this psuedo-operation is entered. The symbolic operation, if not a psuedo-operation, is assumed to be a machine instruction; and its

value is evaluated by comparing it to the entries in the machine instructions table. If the symbolic operation is not found in this table, an error message is typed out and the value of operation code is assumed to be zero. After having evaluated the operation code, the variable field is processed. The variable field may either be blank or contain an address. If the variable field is not blank, the address is evaluated. Zero address is assumed for the blank variable field. Now the address is checked to see which sector it refers to. If it refers to sector zero, sector bit is set to zero and address left unaltered, if it does not, then the sector bit is set to one and the sector number of the address is compared with the sector number to which location counter is set. If both are not same an error message is typed out. The sector number is removed from the address portion and operation code, sector bit, and truncated address are all combined by a logical 'OR' operation to form one 12-bit machine word. The machine word is output into the object deck. The location counter is increased by one and the next statement is read. The process continues until psuedo-operation END is encountered.

The psuedo-operations are handled as special cases, as they were in pass one, by transferring control to a special routine associated with each psuedo-operation. These routines re-enter the control routine of pass two at strategic points. All the psuedo-operations, with the exception of EQU, require the repetition of all the tasks performed in pass one (except symbol definition) for their processing. In addition, data generation psuedo-operations require assembling data in the binary form and including these data words into the object deck. The DEC, OCT, and DSA are handled alike except that evaluation of data word in each case is different. OCT

contains an octal constant, DEC contains a decimal constant, and DSA may contain a relative address. Processing for BCI is slightly different in that it may generate more than one data words. EQU psuedo-operation is ignored. Finally, when END is encountered the address field is evaluated, entered into object deck as entry point for the object program, and pass two terminated.

4.4 Structure of the TAL Assembler

The subroutines and tables that constitute the assembly program TAL have been divided into the following four groups:

1. Control Routines: these determine the logic of the assembly operation.
2. Functional Subroutines: these perform the major functions of the assembler like punching object program, assigning value to a symbol, and evaluating the address portion.
3. Utility Subroutines: these include input-output routines, data conversion routines, a general table search routine, and an error message routine.
4. Assembler Tables: this group includes machine instructions table, psuedo-operations table, and symbol table.

In the structural hierarchy at the top are the control routines, then the functional routines followed by utility routines. Tables can be referred to by any routines but can be manipulated only by utility routines. As would be evident the assembler is highly modular in structure. Each subroutine performs a specific function. All the subroutines are independent

as much as possible and obey a uniform set of conventions regarding inter-communication and interaction. This modular approach reduces programming efforts and makes it easy to cope with additions and modifications.

4.4.1 Control Routines

There are two major routines, PASS 1 and PASS 2, which, as the name suggests control the assembly process during both the passes. A third group that contains one routine for each psuedo-operation, is used to process psuedo-operations. Although these routines have been separated out for the sake of clarity, they are an integral part of the routines PASS 1 and PASS 2, and communicate with them at strategic points. Each psuedo-operation has one entry in the psuedo-operations table. The 'value' of the entry is the address of the routine associated with it. Whenever a psuedo-operation is encountered the corresponding routine is entered. In processing the psuedo-operations, these routines make use of a one word switch, called PASS, which tells them whether entry is from PASS 1 or PASS 2. The switch PASS is located in the sector zero, and is set to zero by PASS 1. PASS 2 sets the switch to one. Since almost every psuedo-operation processing routine performs many functions common to both passes, this structuring reduces the amount of coding. Also in this structure adding new psuedo-operations is extremely simple. To add a new psuedo-operation one has to make an entry into the table and include a routine to process this psuedo-operation. No other portions are effected. All the major functions of the assembly are performed using the functional and utility subroutines. The only exceptions are the manipulation of the location counter and assembling of various portions of a machine instruction. A rather small

number of instructions is incorporated perform these functions.

Location counter is located in the sector zero, with the name LC. The contents of LC can be used by all the routines but can be manipulated (only and) only by the control routines. Utility routines never refer to LC.

The calling sequences for PASS 1 and PASS 2 are as following:

```
PASS 1:     JMS*    *+1
            DSA     PASS 1

PASS 2:     JMS*    *+1
            DSA     PASS 2
```

4.4.2 Functional Routines

There are three major subroutines that correspond to the three major functions of the assembler: (1) assigning a value to the symbol in the name field, (2) Evaluating the address portion of an instruction, and (3) punching the object deck. It is not that other functions are of any less importance but that the most of the other functions, like searching through machine operations table, and data conversion, can be performed directly by the utility routines and hence no corresponding routines are included in this group. The two other major functions i.e. assembly of the various portions of the instruction and counting storage locations are performed by the control routines themselves. Following is a short description of each of these routines along with their calling sequences.

AVSL: Assign value to symbol in name (location) field:

Calling Sequence:

```
JMS*    *+1
DSA     AVSL
```

This subroutine assigns the contents of LC as the value of the symbol in the name field. If name field is blank nothing is done. Before assigning the value to the symbol, it makes a search through the symbol table to see if the symbol has already beend efined, in which case, an error message is typed and symbol is not entered into the table.

ESCD: Evaluate Symbol followed by a Decimal Constant

Calling sequence:

```
JMS*    *+1
DSA     ESCD
DSA     LOC
```

This subroutine is used to compute address portion. LOC is the location of the word in which first character of a relative address is placed. The address portion is terminated by a blank. The value of the address is left in AC on return. If address portion contains an undefined symbol or an illegal constant suitable error messages are typed out.

EWOD and EOD: Object Deck Routines

EWOD: Enter Word in the Object Deck

Calling sequence:

```
JMS*    *+1
DSA     EWOD
```

The contents of the location ODPLC are put into the object deck to be stored, at object time, into the location contained in LC.

EOD: Signify end of Object Deck

Calling sequence:

```
JMS*    *+1
DSA     EOD
```

Contents of LC are put into the object deck as the location where object program shall start.

4.4.3 Utility Subroutines

ECD: Evaluate Constant Decimal

Calling sequence:

```
JMS*    *+1
DSA     ECD
DSA     LOC
```

ECD evaluates a signed or unsigned decimal constant whose first character is at LOC. The value is returned to AC.

ECO: Evaluate Constant Octal

Calling sequence:

```
JMS*    *+1
DSA     ECO
DSA     LOC
```

Same as ECD except that the constant is octal. ESCD, ECQ, and ECD all generate an error messages in the case an illegal character is encountered by them.

SEARCH: Search a Table

Calling sequence:

```
JMS*    *+1
DSA     SEARCH
DSA     LOC
DSA     T1
DSA     T2
xxx     xxxx    Symbol not defined return
xxx     xxxx    Symbol found return
```

This subroutine searches a table for a five character symbol, first of which is at LOC. T1 and T2 are the addresses that describe the origin and the end of the table. Ti is the address of the memory word that contains the location of the first entry in the table. Similarily T2 describes the last entry of the table. Format of the tables is described elsewhere in this chapter.

COW4C: Convert an Octal Word into 4 Characters

Calling sequence:

```
JMS*    *+1
DSA     COW4C
DSA     CC
DSA     WORD
```

WORD contains an Octal constant and CC is the location where internal representation of first octal digit is to be placed.

PST: Print Symbol Table

Calling sequence:

```
JMS*    *+1
DSA     PST
```

ERROR: Type an Error Message

Calling sequence:

```
JMS*    *+1
DSA     READ
```

READS a source statement into LINE+1 to LINE+72

4.4.4 Tables of the TAL Assembler

There are three tables in the TAL assembler:

(1) Symbol Table

(2) Machine Instructions Table

(3) Psuedo-Operations Table

All the tables confirm to a generalized format and can be searched by the routine SERCH. Entries are serially arranged and each entry is six words long. First five words contain the symbolic part and sixth word contains the value associated with the corresponding symbol. Affixed to each table there are two words called T1 and T2 words for that table. Ti word contains the address of the first entry in the table and T2 word contains the address of the last entry of the table. Each table can be then completely specified by the address pair giving its T1 and T2 word addresses. Symbol table at present can accommodate only hundred symbols.

REFERENCES

1. Shirgal R. (1968)

   Simulation of the TDC-12 on IBM-7044, M.Tech. Thesis
   Indian Institute of Technology, Kanpur

2. Wattenburg H.W. (1966)

   Design Automation for Computer Software
   IEEE Transactions on Electronic Computers, Vol.15, no.3,
   pp: 378-380

3. Worthington J.H. (1964)

   The Anatomy of an Assembly System
   Computers and Automation, vol. 13, no.1, pp: 18-20

4. Ferguson D.E. (1966)

   Evolution of the Meta-Assembly Program
   Communications of ACM, vol. 9, no. 3, pp: 18-20

5. Carbatto F.J., Poduska J.W., and Saltzer J.H. (1963)

   Advanced Computer Programming
   MIT Press, Cambridge, Massachusetts

6. Clementson A.J. (1967)

   An Assembly and Loading System for Computers with
   Parallel Peripheral Operation, Computer Journal, vol.9,
   no.4, pp: 370-372

7. Flores I. (1965)

   Computer Software
   Prentice Hall Inc., Englewood Cliffs, N.J.

8. Harrison M.C., and Schwartz J.T. (1967)

   SHARER, A Time Shearing System for the CDC6600
   Communications of ACM, vol.10, no.10, pp: 659-665

CHAPTER FIVE

CONCLUSIONS

TAL assembler at present, is a two pass assembly program. Some of the important features are:

1. Symbolic relative and self relative addressing system. Symbols may be declared any where in the program.
2. Text facility: a contiguous string of characters can be declared in one single psuedo-instruction.
3. Data declaration: octal and decimal data items can be included in the object program by using psedo-opcodes.
4. Symbol definition: a symbol may be assigned a value relative to any other symbol, which has previously been defined.
5. Storage assignment: by using location counter psuedo-opcodes a programmer can assign any section of the program to any desired location.

The assembler as such can not be directly executed on the TDC-12 for two reasons: (1) No specification was available about input-output to the TDC-12, except that it will initially have only two teletype units with paper tape mechanism, and (2) TDC-12 will initially have only paper tape input-output and none of it was available on IBM 7044. Certain assumptions regarding addressing of units and internal representation of characters had to be made to complete the assembler. Details can be found in the reference 1. Incorporating changes in TAL44 and TAL when the input-output specifications become available would be very easy.

The present version of the assembler TAL is by no means complete. It should be considered only as a first version. The structure of the assembler is such that it will be easy to make improvements. There is need for improvements in the following three directions:

1. Permitting arithmetic expressions in the address portion.
2. Data declaration psuedo-operations should be modified to permit more than one item per statement.
3. The ability to call and define macros.

First two improvements require that (1) ECD and ECO routine be modified to recognise ',', ' ', '-', '*', and '/' in addition to blank as terminating a constant, and (2) ESCD routine be replaced by an expression evaluation routine. Changes in psuedo-opcode routines DEC, OCT, and DSA will be trivial. Inclusion of Macro facility will, however, require much more effort. Two additional routines, one to process Macro-definitions and the other to expand the Macro-instructions would be required. A Macro-definition psuedo-operation should be included in psuedo-operations table. The corresponding routine will make entries in two tables when a Macro-defintion is encountered. The first table, called Macro-operations table will contain each symbolic opcode defined as a Macro. The second table called Macro-definitions table will contain Macro-definitions. The location from which a Macro-definition starts, will be put in the Macro-operations table as 'value' associated the corresponding entry. The main processing loops of pass one and pass two, before checking for a psuedo-operation, would check the symbolic-operation to see if it is included in Macro-operations table. If it is Macro-operation control, the control will be

transferred to the Macro-expanding routine. This routine will generate the card images and pass them on two main routines of pass one and pass two as the case may be. For nested and recursive Macros the main bodies of Macro-definition, pass one and pass two routines will have to be recursive. This can be taken care of by a simple push down stack.

Inclusion of user defined Macro-operations, in the present framework of the assembler may be an interesting and challenging exercise. It must be emphasised here that there is a strong need for preserving the modularity of the TAL assembler while making any changes. That, Macros can be included in the present framework of the assembler is a strong argument in the case. TAl assembler has been tested and debugged on the TDC-12 simulator on IBM 7044.

APPENDIX-1

IMPLEMENTATION OF TAL ASSEMBLER ON TDC-12

Since no information was available regarding the input-output on the TDC-12, certain assumptions had to be made in writing the TAL and TAL44 assemblers. Details of these assumptions are given in the reference 1. The present version of the TAL assembler, therefore, can not be executed on the TDC-12, unless necessary changes have been made to conform to the TDC-12 characteristics.

The TAL-44 assembler should be changed in following respects:

1. In machine instructions table, the numeric opcodes corresponding to symbolic I/O instructions, need to be replaced by actual opcodes.

2. The BCI psuedo-operation generates one binary word for each character. The code that is generated for each character is taken from a table included in the routine T.BCI. This table should be changed.

Following changes should be made in the TAL assembler:

1. In machine instructions table, replace the numeric opcodes corresponding to the symbolic I/O instructions by their actual values.

2. Internal representation for carriage return and line feed are generated using their octal values. Declarations for both these characters appear at the end in the source program for TAL assembler. The octal values should be replaced by the corresponding actual values.

3. Present version of the TAL assembler, when executed on TDC-12 will produce an object deck in BCD format. In this format each

binary word is represented by four octal digits and hence each word on object deck occupies four characters. This is very inefficient. Much better and efficient use of paper tape can be made, if object deck is punched in binary form in which each word can be represented by two characters on the paper tape. Since no information regarding binary I/O was available it could not be implemented in the present version of the TAL assembler. Since no format for the object program that would make efficient use of the paper tape I/O, could be decided it was not possible to write a binary loader. No information was available about getting the machine initially started. How to enter instructions into core storage that will bootstrap a loader is still not clear. This was another difficulty that prevented writing of a loader. To obtain a version of the TAL assembler that can be executed on the TDC-12 therefore, it is necessary to write a loader and then punch an object tape according to the format specified by the loader. The TAL 44 assembler operating on the IBM 7044, will produce an assembly listing, when TAL assembler is input to this. Object program for TAL assembler can be obtained from this listing.

APPENDIX TWO

TDC-12 INSTRUCTION REPERTOIRE

Storage Reference Instructions

Mnemonic	Octal Code	Operation
AND Y	04	Logical AND
XOR Y		Exclusive OR
LAC Y	14	Load Acuumulator
SAC Y	20	Store Accumulator
ADD Y	24	Add to the Accumulator
SUB Y	30	Subtract from the Accumulator
RAD Y	34	Replace add memory
ISZ Y	40	Increment storage by one and skip if zero
JMP Y	44	Jump to Y
JMS Y	50	Jump to subroutine Y
CAS Y	54	Compare Accumulator and skip
XCT Y	60	Execute instruction at location Y

Non Storage Reference Instructions

SMK	6401	Set mask flip flops
ION	6402	Turn interrupt on
IOF	6404	Turn interrupt off
NOP	7000	No operation
CLA	7001	Clear Accumulator
CLC	7002	Clear Carry regester
CAR	7010	Circulate accumulator right
CAL	7020	Circulate AC left

Mnemonic	Octal Code	Operation
CTR	7014	Circulate AC two right
CTL	7024	Circulate AC two left
CMC	7040	Complement Carry register
CMA	7100	Complement AC
IAC	7200	Increment AC by 1
CIA	7300	Complement and increment AC
STC	7042	Set Carry Register
STA	7101	Set AC
CLA	7401	Clear AC
ORS	7402	OR with switch register
SKP	7404	Unconditional Skip
SNC	7410	Skip on non-zero carry register
SZC	7414	Skip on zero carry register
SZ A	7420	Skip on zero AC
SNA	7424	Skip on non zero AC
SMA	7440	Skip on minus AC
SPA	7444	Skip on positive accumulator
STP	7500	STOP

```
ETAL44 TALC
*****************************************************************************
       REM    MAIN PROGRAM
*****************************************************************************
       ORG    64
START  JMS*   *+1
       DSA    PASS1
       JMS*   *+1
       DSA    PASS2
       JMS*   *+1
       DSA    PST
       STP
*****************************************************************************

       EJECT
```

```
*****************************************************************************
      REM   PASS2
*****************************************************************************
      REM   ENTRANCE..  JMS*  *+1
      REM               DSA   PASS2
*****************************************************************************
      OGS
ASS2  NOP
      ISZ   PASS2       SET RETURN ADRESS
      LAC   ONE         INITIALIZE PASS2
      SAC   PASS
      CLA
      SAC   LC          INITIALIZE LOCATION COUNTER
      SAC   CC          INITIALIZE CARD  COUNT
2001  JMS*  *+1         READ CARD
      DSA   READ
      ISZ   CC
      LAC*  P2012       IS IT * CARD
      CAS   P2006
      JMP   *+2         NO
      JMP   P2002       YES
      JMS*  *+1         IS IT PSUEDO OPCODE
      DSA   SERCH
      DSA   OF
      DSA   POPT1
      DSA   POPT2
      JMP   *+4         NO
      SAC   *+2         YES. VALUE IN ACC
      JMP*  *+1         TRANSFER TO PSUEDO OPCODE PROCESSING ROUTINE
      DEC   0
      JMS*  *+1
      DSA   SERCH
      DSA   OF
      DSA   OPT1
      DSA   OPT2
      JMP*  P2015       NO. ERROR
2003  SAC   P2007       YES
      JMS*  *+1         EVALUATE VARIABLE FIELD
      DSA   ESCD
      DSA   VF
      SAC   P2008
      AND   P2009
      SZA               IS SECTOR ZERO
      JMP   *+3         NO
      LAC   P2008       YES
      JMP   P2004
      SAC   P2010       NO
      LAC   LC
      AND   P2009
      CAS   P2010       SAME SECTOR
```

```
        JMP    *+2
        JMP    P2005          YES
        JMS*   ERROR          NO
        OCT    4
P2005   LAC    P2008          LOAD ADRESS
        AND    P2013
        ADD    P2014
P2004   ADD    P2007
P20BB   SAC    AI
        JMS*   *+1            SEND TO OBJECT DECK
        DSA    EWOD
P20AA   ISZ    LC
P2002   JMP    P2001
        REM
*************************************************************************
P2006   BCI    1 *
P2009   OCT    7700
P2012   DSA    LF
P2013   OCT    0077
P2014   OCT    0100
P2015   DSA    P2ER1
P2007   BSS    1              OPCODE
P2008   BSS    1              ADRESS PORTION
*************************************************************************
        REM
        OGS
P2ER1   JMS*   ERROR
        DEC    5
        CLA
        JMP*   *+1
        DSA    P2003
*************************************************************************
*
        EJECT
```

```
*************************************************************************
      REM   END           PUSEDO OP CODE END
*************************************************************************
END   LAC   PASS
      SNA
      JMP   END01         PASS1
      JMS*  *+1           PASS2, EVALUATE VARIABLE FIELD
      DSA   ESCD
      DSA   VF
      SAC   LC            SAVE ENTRY POINT IN LC
      JMS*  *+1           SIGNIFY END OF OBJECT DECK
      DSA   EOD
      LAC*  END04
      JMP   END02
END01 LAC*  END03
END02 SAC   END05
      JMP*  END05
      REM
*************************************************************************
END03 DSA   PASS1
END04 DSA   PASS2
END05 DEC   00            TEMP
*************************************************************************
*
      EJECT
```

```
*************************************************************************
      REM   ORG          PSUEDO OP CODE ORG
*************************************************************************
      OGS
ORG   NOP
      JMS*  *+1          EVALUATE VARIABLE FIELD
      DSA   ESCD
      DSA   VF
ORG02 SAC   LC
      LAC   PASS
      SZA
      JMP*  P2I02        PASS2
      JMS*  *+1          PASS1, ASSIGN VALUE TO SYMBOL
      DSA   AVSL
      JMP*  P1I02
*************************************************************************
*
      EJECT
```

```
*************************************************************************
      REM   OGS          PSUEDO OPCODE OGS
*************************************************************************
OGS   LAC   LC
      AND   OGS03
      SNA                IS IT BEGINNING OF A NEW SECTOR
      JMP   OGS01        YES
      LAC   LC
      AND   OGS04
      ADD   OGS05
      JMP   ORG02
OGS01 LAC   LC
      JMP   ORG02
      REM
*************************************************************************
P2I02 DSA   P2002
P2I01 DSA   P2001
P1I02 DSA   P1002
OGS03 OCT   0077
OGS04 OCT   7700
OGS05 OCT   0100
*************************************************************************
*
      EJECT
```

```
**************************************************************************
      REM   EJCT          PSUEDO OPCODE REM AND EJECT
**************************************************************************
EJCT  EQU   *
REM   LAC   PASS
      SNA
      JMP*  P1I02         PASS1
      JMP*  P2I01         PASS2
**************************************************************************
*
      EJECT
```

```
*************************************************************************
      REM   EQU          PSUEDO OPCODE EQU
*************************************************************************
EQU   JMS*  *+1          EVALUATE VARIABLE FIELD
      DSA   ESCD
      DSA   VF
      SAC   EQU02
      LAC   PASS
      SZA
      JMP*  P2I02        PASS2
      LAC   LC           PASS 1  SAVE LC
      SAC   EQU03
      LAC   EQU02
      SAC   LC
      JMS*  *+1
      DSA   AVSL
      LAC   EQU03        RESTORE LC
      SAC   LC
      JMP*  P1I02
      REM
*************************************************************************
EQU02 DEC   00
EQU03 DEC   00           SAVE LC IN THIS
*************************************************************************
*
      EJECT
```

```
*************************************************************************
      REM   BSS          PSUEDO OPCODE BSS
*************************************************************************
BSS   LAC   PASS
      SZA
      JMP   *+3          PASS2
      JMS*  *+1          PASS1
      DSA   AVSL
      JMS*  *+1
      DSA   ECD
      DSA   VF
      RAD   LC
      LAC   PASS
      SNA
      JMP*  P1I02        PASS1
      JMP*  P2I02        PASS2
*************************************************************************
*
      EJECT
```

```
**************************************************************************
       REM   OCT          PSUEDO OPCODE OCT
**************************************************************************
       OGS
OCT    LAC   PASS
       SNA
       JMP*  01101          PASS1
       JMS*  *+1            PASS2
       DSA   ECO            EVALUATE OCTAL CONSTANT
       DSA   VF
       JMP*  02IBB
**************************************************************************
*
       REM
       REM
       REM
**************************************************************************
DEC    LAC   PASS
       SNA
       JMP*  01101
       JMS*  *+1
       DSA   ECD
       DSA   VF
       JMP*  02IBB
**************************************************************************
*
       REM
       REM
**************************************************************************
DSA    LAC   PASS
       SNA
       JMP*  01101
       JMS*  *+1
       DSA   ESCD
       DSA   VF
       JMP*  02IBB
       REM
**************************************************************************
02IBB  DSA   P20BB
01101  DSA   P1001
**************************************************************************
*
       EJECT
```

```
*************************************************************************
      REM   BCI           PSUEDO OPCODE BCI
*************************************************************************
BCI   LAC   'PASS
      SNA
      JMP*  BCI08         PASS1
      JMS*  *+1           PASS2
      DSA   ECD           EVALUATE CONSTANT
      DSA   VF
      CIA                 SET COUNTER
      SAC   BCI04
      LAC   BCI09
      SAC   BCI8
      LAC*  BCI8
      SZA
      JMP   *-2
BCI02 LAC*  BCI8          GET CHARACTER
      SAC   AI
      JMS*  *+1           SEND TO OBJECT DECK
      DSA   EWOD
      ISZ   LC
      ISZ   BCI04
      JMP   BCI02
      JMP*  BCI05         P2001
      REM
*************************************************************************
BCI8  EQU   8
BCI05 DSA   P2001
BCI08 DSA   BSS
BCI09 DSA   VF-1
BCI04 BSS   1             VALUE OF CONSTANT
*************************************************************************
*
      EJECT
```

```
*************************************************************************
      REM   AVSL          ASSIGN VALUE TO SYMBOL IN LOCATION FIELD
*************************************************************************
      REM   ENTRANCE..    JMS*  *+1
      REM                 DSA  AVSL
*************************************************************************
      OGS
AVSL  NOP
      ISZ   AVSL
      LAC*  AVSL2         IS IT BLANK
      CAS   BLANK
      JMP   *+3           NO
      JMP*  AVSL          YES .  RETURN
      NOP                 NO
      JMS*  *+1           IS SYMBOL ALREADY DEFINED
      DSA   SERCH
      DSA   LF
      DSA   STT1
      DSA   STT2
      JMP   AVSL1         NO
      JMS*  ERROR         YES= ERROR 1
      DEC   1
      JMP*  AVSL
      REM
AVSL1 JMS*  *+1           DEFINE
      DSA   ESEST
AVSL2 DSA   LF
      DSA   LC
      JMP*  AVSL          RETURN
*************************************************************************
*
      EJECT
```

```
**********************************************************************
      REM   ESCD          EVALUATE SYMBOL AND COMSTANT
**********************************************************************
      REM   ENTRANCE..    JMS*  *+1
      REM                 DSA   ESCD
      REM                 DSA   LOC
      REM   OUTPUT..      VALUE IN AC
**********************************************************************
      OGS
ESCD  NOP
      CLA
      SAC   ESCDE         VALUE OF SYMBOL ZERO
      ISZ   ESCD
      LAC*  ESCD
      SUB   ONE
      SAC   ESCD8         CHARACTER LOCATION
      ISZ   ESCD          RETURN ADRESS
      LAC   ESCDJ
      SAC   ESCD9         SYMBOL LOCATION
      LAC   ESCDA
      SAC   ESCDB         SET LOCATION COUNTER TO FOUR
      LAC*  ESCD8         CHARACTER TO ACC
      CAS   A             IS IT ONE OF THE ENGLISH ALPHABET
      JMP   ESCD2         YES
      JMP   ESCD2         YES
      JMP*  ESC11         NO. GO TO NO SYMBOL
ESCD1 LAC*  ESCD8
      CAS   A             IS IT ONE OF THE ENGLISH ALPHABET
      JMP   ESCD2         YES
      JMP   ESCD2         YES
      CAS   ZERO          NO.  IS IT GREATER THAN ZERO
      JMP   *+3           YES
      JMP   *+2           YES
      JMP   ESCD3         NO. SYMBOL LESS THAN FIVE CHARACTERS
      CAS   NINE          IS IT LESS THAN 9
      JMP   ESCD3         NO. SYMBOL LESS THAN FIVE CHARACTERS
      JMP   ESCD2         YES
      JMP   ESCD2         YES
ESCD2 SAC*  ESCD9
      ISZ   ESCDB         FIVE CHARACTERS OVER
      JMP   ESCD1         NO
      LAC   ESCD8         YES SET CONSTANT ADRESS
      ADD   ONE
      SAC*  ESCD4
      JMP*  ESCD5
ESCD3 CLA                 SYMBOL LESS THAN FIVE CHARACTERS
      SAC*  ESCD9
      ISZ   ESCDB
      JMP   ESCD3+1
      LAC   ESCD8
```

```
        SAC*   ESCD4
        JMP*   ESCD5
        REM
*****************************************************************
ESC11   DSA    ESC12          NO SYMBOL EROUTINE ADRESS
ESCD4   DSA    ESCD7          CONSTANT ADRESS
ESCD5   DSA    ESCD6          ADRESS TO GET VALUE OF SYMBOL
ESCD8   EQU    8
ESCD9   EQU    9
ESCDA   DEC    -5
ESCDJ   DSA    *
        BSS    5
ESCDB   BSS    1              COUNTER
ESCDE   BSS    1              VALUE OF SYMBOL
        OGS
ESCD6   JMS*   *+1            GET VALUE OF SYMBOL
        DSA    SERCH
        DSA    ESCDJ+1
        DSA    STT1
        DSA    STT2
        JMP    ESCER          UNDEFINED
ESC8    SAC*   ESCE           DEFINED
        JMS*   *+1            GET CONSTANT
        DSA    ECD
ESCD7   DSA    00             TO BE SET
        ADD*   ESCE           GET VALUE
        SAC    ESC13          SAVE VALUE
        LAC*   ESC            FORM RETURN ADRESS
        SAC    *+3
        LAC    ESC13          VALUE TO ACC
        JMP*   *+1            RETURN
        DSA    00
        REM
        REM
ESCER   JMS*   ERROR
        OCT    3              UNDEFINED SYMBOL
        CLA
        JMP    ESC8
ESC12   CAS    STAR           NO SYMBOL.  IS IT STAR
        JMP    *+2            NO
        JMP    ESC14          YES
        LAC    ESCD8          NO
        SAC    ESCD7
        JMP    ESC8+1
ESC14   LAC    ESCD8
        ADD    ONE
        SAC    ESCD7
        LAC    LC
        JMP    ESC8
        REM
```

```
**************************************************************************
ESCE  DSA   ESCDE
ESC   DSA   ESCD
ESC13 DSA   00
**************************************************************************
*
      EJECT
```

```
*****************************************************************************
       REM   EDC          EVALUATE DECIMAL CONSTANT
*****************************************************************************
       REM   ENTRANCE..   JMS*  *+1
       REM                DSA   ECD
       REM                DSA   LOC
       REM   INPUT..      FROM LOC ONWARDS A SIGNED OT UNSIGNED INTEGER
       REM   OUTPUT..     VALUE IN ACCUMULATOR
*****************************************************************************
       OGS
ECD    NOP
       CLA
       SAC   ECD08        VALUE ZERO
       SAC   ECD09
       LAC   ECDM4        COUNTER FOR FOUR DIGITS
       SAC   ECD10
       ISZ   ECD
       LAC*  ECD
       ISZ   ECD          SET RETURN ADRESS
       SUB   ONE          FORM CHARACTER ADRESS
       SAC   ECD8
       LAC*  ECD8         FIRST CHARACTER
       CAS   PLUS         IS IT +
       JMP   *+2          NO
       JMP   ECD02        YES +
       CAS   MINUS        NO.  IS IT -
       JMP   *+2          NO
       JMP   ECD01        YES -
       JMP   ECD03        NO
ECD01  LAC   ONE          MINUS
       CIA                -1 IN ACC
       SAC   ECD09        SET SWITCH FOR MINUS
ECD02  LAC*  ECD8         NEXT CHARACTER
ECD03  SNA                IS IT BLANK
       JMP   ECD07        YES
       CAS   ZERO         NO IS IT GREATER THAN 0
       JMP   *+3          YES
       JMP   *+2          YES
       JMP   ECDER        NO
ECD04  CAS   NINE         IS IT LESS THAN NINE
       JMP   ECDER        NO
       NOP                YES
       SUB   ZERO         YES. GET DIGIT
       SAC   ECD11        STORE DIGIT
       LAC   ECD08        VALUE TO ACC
       CTL
       CAL
       AND   ECD12
ECD05  ADD   ECD08
       ADD   ECD08        MULTIPLIED BY TEN
```

```
       ADD    ECD11          ADD DIGIT
       SAC    ECD08          STORE VALUE
       ISZ    ECD10          FOUR CHARACTERS OBER
       JMP    ECD02          NO
ECD07  LAC    ECD08          VALUE TO ACC
       ISZ    ECD09          IS IT MINUS
       JMP    *+2            NO
       CIA                   YES
       JMP*   ECD
ECDER  JMS*   ERROR
       DEC    2              ILLEGAL CONSTANT
       CLA                   ZERO VALUE
       JMP*   ECD            RETURN
       REM
*************************************************************************
ECD12  OCT    7770
ECD8   EQU    8
ECDM4  DEC    -4
ECD08  BSS    1              VALUE
ECD09  BSS    1              0 FOR +, -1 FOR -VE
ECD10  BSS    1              COUNTER
ECD11  BSS    1              DIGIT
*************************************************************************
*
       EJECT
```

```
*************************************************************************
      REM'  ECO          EVALUATE OCTAL CONSTANT
*************************************************************************
      REM   ENTRANCE..   JMS*  **+1
      REM                DSA   ECO
      REM                DSA   LOC
      REM   INPUT.. OUTPUT.. SAME AS ECD
*************************************************************************
      OGS
ECO   NOP
      LAC   ECO03
      SAC*  ECO04
      LAC   ECO06
      SAC*  ECO05
      ISZ   ECO
      LAC*  ECO
      SAC   ECO01
      JMS*  **+1
      DSA   ECO
ECO01 DEC   00
      SAC   ECO07
      LAC   ECO02
      SAC*  ECO04
      LAC*  ECO08
      SAC*  ECO05
      ISZ   ECO
      LAC   ECO07
      JMP*  ECO
ECO02 CAS   NINE
ECO03 CAS   SEVEN
ECO04 DSA   ECO04
ECO05 DSA   ECO05
ECO06 SKP
ECO08 DSA   ECO05+1
ECO07 BSS   1
*************************************************************************
*
      EJECT
```

```
*****************************************************************************
       REM   SERCH        SERCH A TABLE
*****************************************************************************
       REM   ENTRANCE..   JMS*  *+1
       REM                DSA   SERCH
       REM                DSA   SYMBOL LOC
       REM                DSA   T1 T1 ADRESS
       REM                DSA   T2 T2 ADRESS
       REM                ***   ****    UNDEFINED
       REM                ***   ****    DEFINED
       REM   INPUT..      FIVE CHARACTER SYMBOL AT SYMBOL TO SYMBOL+4
       REM                T1  LOCATION THAT CONTAINS BEGINNING
       REM                ADRESS OF TABLE
       REM   T2      LOCATION THAT CONTAINS THE ADRESS OF
       REM           LAST ENTRY IN THE TABLE
       REM                TABLE IS SIX WORD ENTRY.
       REM   OUTPUT..     VALUE OF SYMBOL IF FOUND IN ACC
*****************************************************************************
       OGS
SERCH  NOP
SST    EQU   SERCH
       ISZ   SST          FROM ADRESSS YMBOL-1
       LAC*  SST
       SUB   ONE
       SAC   SST4
       ISZ   SST
       LAC*  SST          T1 ADRESS TO AC
       SAC   SSTEM        ADRESS OF FIRST ENTRY
       LAC*  SSTEM        IN THE TABLE TO ACC
       SUB   ONE
       SAC   SST5         CURRENT ADRESS -1
       ISZ   SST
       LAC*  SST          T2 ADRESS TO ACC
       SAC   SSTEM
       LAC*  SSTEM        ADRESS OF LAST ENTRY TO ACC
       SUB   ONE
       SAC   SST6         END ADEESS-1
       ISZ   SST          SET TO UNDEFINED RETURN
SST1   LAC   SST6
       CAS   SST5         IS IT END OF TABLE
       JMP   *+3          NO
       JMP   *+2        N O
       JMP   SUND         YES. SYMBOL UNDEFINED
       LAC   DM5          LOAD CHARACTER COUNT
       SAC   STCNT
       LAC   SST4         SET TO FRRST CHARACTER
       SAC   SYMBL        MINUS ONE OF THE SYMBOL.
       LAC   SST5         SET TO FIRST CHSRACTER -1 OF THE ENTRY IN TAB
       SAC   STC
       ADD   SIX          PREPARE FOR NEXT ROUND
```

```
        SAC    SST5
SST2    LAC*   SYMBL
        CAS*   STC
        JMP    SST1          TRY NEXT SYMBOL
        JMP    *+2           THIS CHARACTER COMPARES
        JMP    SST1          TRY NEXT SYMBOL
        ISZ    STCNT         ALL CHARACTERS COMPARED
        JMP    SST2          NO. NEXT CHARACTER
        LAC*   STC           YES .  SYMBOL FOUND
        ISZ    SST           VALUE TO ACC.
        JMP*   SST           RETURN
        REM
SUND    CLA                  SYMBOL UNDEFINED
        JMP*   SST
        REM
*************************************************************************
STC     EQU    9             CURRENT CHARACTER INDEX OF TABLE
SYMBL   EQU    8             CURRENT CHARACTER INDEX FOR SYMBOL
SIX     DEC    6
DM5     DEC    -5
SST4    BSS    1             ADRESS OF SYMBOL -1
SST5    BSS    1             CURRENT ADRESS -1
SST6    BSS    1             TABLE LAST ENTRY ADRESS -1
STCNT   BSS    1             CHARACTER COUNT
SSTEM   EQU    SST6          TEMPORARY
*************************************************************************
*
        EJECT
```

```
*****************************************************************************
       REM   ESEST        ENTER A SYMBOL IN SYMBOL TABLE
       REM   ENTRANCE..   JMS*  *+1
       REM                DSA   ESEST
       REM                DSA   SYMBOL LOCATION
       REM                DSA   VALUE LOCATION
       REM   INPUT..      FIVE CHARACTER SYMBOL AND VALUE
*****************************************************************************
       OGS
ESEST  NOP
       LAC   STT2         FORM CURRENT ADRESS OF TABLE
       ADD   ESED5        PLUS FIVE
       SAC   ESES8
       IAC
       CAS   STT3         IS THERE ANY SPACE LEFT IN SYMBOL TABLE.
       JMP   ESEER        NO
       NOP                YES
       SAC   STT2         YES
       ISZ   ESEST        FORM SYMBOL MINUS 1
       LAC*  ESEST
       SUB   ONE
       SAC   ESES9
       LAC   ESEM5        COUNTER
       SAC   ESES5
       LAC*  ESES9        CHARACTER TO TABLE
       SAC*  ESES8
       ISZ   ESES5        ALL MOVED
       JMP   *-3          NO
       ISZ   ESEST        YES
       LAC*  ESEST        VALUE ADRESS TO ACC
       SAC   ESES5
       LAC*  ESES5
       SAC*  ESES8        VALUE TO TABLR
       ISZ   ESEST
       JMP*  ESEST        RETURN
ESEER  JMS*  ERROR
       DEC   5            SYMBOL TABLE FULL
       STP
       REM
*****************************************************************************
ESES8  EQU   8            TABLE
ESES9  EQU   9            SYMBOL
ESED5  DEC   5
ESEM5  DEC   -5
STT3   DSA   STEND
*****************************************************************************
STT1   DSA   STBGN
STT2   DSA   STBGN-6
ESES5  BSS   1            COUNTER
STBGN  BSS   1194         SYMBOL TABLE
```

```
STEND BSS   6               200 SYMBOLS
**************************************************************************
#
        EJECT
```

```
**********************************************************************
      REM   PST          PRINT SYMBOL TABLE
**********************************************************************
      REM   ENTRANCE..   JMS*  *+1
      REM                DSA   PST
**********************************************************************
      OGS
PST   BSS   1
      ISZ   PST
      LAC   PST10
      SUB   ONE
      SAC   PST8         CURRENT TABLE ADRESS
      LAC*  PST11
      SUB   ONE
      SAC   PST12        END ADRESS
PST00 LAC   PST8         HAVE ALL ENTRIES BEEN PRINTED
      CAS   PST12
      JMP*  PST          YES. RETURN
      NOP                NO
      LAC   CR           NO. CARRIAGE RETURN
      TSF
      JMP   *-1
      TLS
      LAC   PST14        PRINT SYMBOL
      SAC   PST15
PST01 LAC*  PST8         NEXT CHARACTER
      TSF
      JMP   *-1
      TLS
      ISZ   PST15
      JMP   PST01
      LAC   PST14        PRINT BLANKS
      SAC   PST15
      CLA
PST02 TSF
      JMP   *-1
      TLS
      ISZ   PST15
      JMP   PST02
      LAC*  PST8         PRINT VALUE
      SAC   PST16
      JMS*  *+1
      DSA   COW4C
      DSA   PST17
      DSA   PST16
      LAC   *-2          PRINT FOUR CHARACTERS
      SUB   ONE          SET CHARACTER ADRESS
      SAC   PST9
      LAC   PST18        SET COUNTER
      SAC   PST15
```

```
PST03 LAC*   PST9
      TSF
      JMP    *-1
      TLS
      ISZ    PST15
      JMP    PST03          VALUE PRINTED
      JMP    PST00          NEXT SYMBOL
      REM
*************************************************************************
PST8  EQU    8              CURRENT TABLE ADRESS GOES HERE
PST9  EQU    9
PST10 DSA    STBGN
PST11 DSA    STT2
PST14 DEC    -5
PST18 DEC    -4
PST12 BSS    1              WILL CONTAIN STEND
PST15 BSS    1              COUNTER
PST16 BSS    1              VALUE
PST17 BSS    4              CHARACTERS FOR VALUE
*************************************************************************
*
      EJECT
```

```
**********************************************************************
       REM   EWOD          ENTER WORD IN OBJECT DECK
**********************************************************************
       REM   ENTRANCE..    JMS*  *+1
       REM                 DSA   EWOD
**********************************************************************
       OGS
EWOD   BSS   1
       ISZ   EWOD          RETURN ADRESS
       LAC   ODRWC         IS WC EQUAL TO ZERO
       SZA
       JMP   EWOD1         NO
EWODO  CLA                 YES, OUTPUT BLANK CHARACTER
       TSP
       JMP   *-1
       TLC
       LAC   LC            OUTPUT LC
       SAC   ODPLC
       JMS*  *+1
       DSA   ODR20
       JMP   EWOD2
EWOD1  LAC   ODPLC         IS WC+PLOC=LC
       ADD   ODRWC
       CAS   LC
       JMP   *+2           NO
       JMP   EWOD2         YES
       JMS   ODR10         NO. PUT TRAILER IN OUTPUT
       JMP   EWODO
EWOD2  ISZ   ODRWC
       LAC   AI
       JMS*  *+1           OUTPUT AI
       DSA   ODR20
       LAC   ODRWC
       CAS   EWO14         IS WC = 14
       JMP   EWOD
       JMS   ODR10
       JMP*  EWOD          RETURN
ODR10  NOP                 PUT TRAILER CHARACTER AND SET WC=0
       LAC   CR
       TSP
       JMP   *-1
       TLC
       CLA
       SAC   ODRWC
       JMP*  ODR10
EWO14  DEC   14
ODRWC  DEC   0             WORD COUNT
ODPLC  DEC   00
**********************************************************************
*
```

** TAL ASSEMBLER LISTINGS **

?EJECT

```
*****************************************************************************
      REM   EOD           ENTER ADRESS AND SIGNIFY END OF DECK
*****************************************************************************
      REM   ENTRANCE..    JMS*  *+1
      REM                 DSA   EOD
*****************************************************************************
EOD   NOP
      ISZ   EOD
      LAC   ODRWC         IS WC EQUAL TO ZERO
      SZA
      JMS   ODR10         NO
      LAC   ONECH
      TSP
      JMP   *-1
      TLC
      LAC   LC
      JMS*  *+1
      DSA   ODR20
      JMS   ODR10
      JMP*  EOD
      REM                 OUTPUT A WORD ON TAPE AS FOUR OCTAL DIGITS
      REM   ENTRANCE..    JMS*  *+1
      REM                 DSA   ODR20
      OGS
ODR20 BSS   1
      SAC   ODR21
      JMS*  *+1
      DSA   COW4C
      DSA   ODR21+1
      DSA   ODR21
      REM   INPUT..       WORD IN ACCUMULATOR
      LAC   ODR22
      SAC   ODR21         SET COUNTER
      LAC   ODR23         SET CHARACTER ADRESS
      SAC   ODR9
ODR25 LAC*  ODR9
      TSP                 PUNCH CAHARACTER
      JMP   *-1
      TLC
      ISZ   ODR21
      JMP   ODR25
      ISZ   ODR20
      JMP*  ODR20
      REM
*****************************************************************************
ODR9  EQU   9
ODR22 DEC   -4
ODR23 DSA   ODR21
ODR21 BSS   5               WORD AND CHARACTER
*****************************************************************************
```

** TAL ASSEMBLER LISTINGS **

EJECT

```
*************************************************************************
      REM   OP CODE TABLE
*************************************************************************
OPBGN EQU   *
      REM
      REM   STORAGE REFERENCE INSTRUCTIONS
      REM
      BCI   5 AND
      OCT   0400
      BCI   5 AND*
      OCT   0600
      BCI   5 XOR
      OCT   1000
      BCI   5 XOR*
      OCT   1200
      BCI   5 LAC
      OCT   1400
      BCI   5 LAC*
      OCT   1600
      BCI   5 SAC
      OCT   2000
      BCI   5 SAC*
      OCT   2200
      BCI   5 ADD
      OCT   2400
      BCI   5 ADD*
      OCT   2600
      BCI   5 SUB
      OCT   3000
      BCI   5 SUB*
      OCT   3200
      BCI   5 RAD
      OCT   3400
      BCI   5 RAD*
      OCT   3600
      BCI   5 ISZ
      OCT   4000
      BCI   5 ISZ*
      OCT   4200
      BCI   5 JMP
      OCT   4400
      BCI   5 JMP*
      OCT   4600
      BCI   5 JMS
      OCT   5000
      BCI   5 JMS*
      OCT   5200
      BCI   5 CAS
      OCT   5400
      BCI   5 CAS*
```

```
OCT   5600
BCI   5 XCT
OCT   6000
BCI   5 XCT*
OCT   6200
REM
REM   REGISTER SET 1
REM
BCI   5 NOP
OCT   7000
BCI   5 CLA
OCT   7001
BCI   5 CLC
OCT   7002
BCI   5 CAR
OCT   7010
BCI   5 CAL
OCT   7020
BCI   5 CTR
OCT   7014
BCI   5 CTL
OCT   7024
BCI   5 CMC
OCT   7040
BCI   5 CMA
OCT   7100
BCI   5 IAC
OCT   7200
BCI   5 CIA
OCT   7300
BCI   5 STC
OCT   7042
BCI   5 STA
OCT   7101
REM
REM   REGISTER SET 2
REM
BCI   5 CLA
OCT   7401
BCI   5 ORS
OCT   7402
BCI   5 SKP
OCT   7404
BCI   5 SNC
OCT   7410
BCI   5 SZC
OCT   7414
BCI   5 SZA
OCT   7420
BCI   5 SNA
```

```
OCT    7424
BCI    5 SMA
OCT    7440
BCI    5 SPA
OCT    7444
BCI    5 STP
OCT    7500
REM
REM    INTERRUPT INSTRUCTIONS
REM
BCI    5 IOF
OCT    6404
BCI    5 ION
OCT    6402
BCI    5 SMK
OCT    6401
BCI    5 SMON
OCT    6403
REM
REM    INPUT OUTPUT INSTRUCTIONS
REM
BCI    5 KSF
OCT    6414
BCI    5 KCC
OCT    6412
BCI    5 KRS
OCT    6411
BCI    5 KRB
OCT    6413
BCI    5 KSP
OCT    6424
BCI    5 KCS
OCT    6422
BCI    5 KRC
OCT    6421
BCI    5 KRP
OCT    6423
BCI    5 TSF
OCT    6434
BCI    5 TCF
OCT    6432
BCI    5 TPC
OCT    6431
BCI    5 TLS
OCT    6433
BCI    5 TSP
OCT    6444
BCI    5 TCP
OCT    6442
BCI    5 TPS
```

```
        OCT    6441
OPEND   EQU    *
        BCI    5 TLC
        OCT    6443
*************************************************************************
OPT1    DSA    OPBGN
OPT2    DSA    OPEND
*************************************************************************
*
        EJECT
```

```
**************************************************************************
      REM   PSUEDO OP CODE TABLE
**************************************************************************
POBGN EQU   *
      BCI   5 DEC
      DSA   DEC
      BCI   5 OCT
      DSA   OCT
      BCI   5 DSA
      DSA   DSA
      BCI   5 BSS
      DSA   BSS
      BCI   5 OGS
      DSA   OGS
      BCI   5 ORG
      DSA   ORG
      BCI   5 EQU
      DSA   EQU
      BCI   5 BCI
      DSA   BCI
      BCI   5 REM
      DSA   REM
      BCI   5 EJECT
      DSA   EJCT
POEND EQU   *
      BCI   5 END
      DSA   END
**************************************************************************
POPT1 DSA   POBGN
POPT2 DSA   POEND
**************************************************************************
*
      EJECT
```

```
**************************************************************************
       REM   COW4C          CONVERT AN OCTAL WORD INTO FOUR CHARACTERS
**************************************************************************
       REM   ENTRANCE..  JMS*  *+1
       REM               DSA   COW4C
       REM               DSA   LOC CHARACTER
       REM               DSA   WORD
**************************************************************************
       OGS
COW4C  BSS   1
       LAC   COW05        SET COUNTER
       SAC   COW06
       ISZ   COW4C
       LAC*  COW4C        ADRESS OF CHARACTERS
       SUB   ONE
       SAC   COW8
       ISZ   COW4C
       LAC*  COW4C
       SAC   COW07
       LAC*  COW07
       SAC   COW07        WORD TO COW07
       ISZ   COW4C        RETURN ADRESS
COW01  LAC   COW07
       CTL
       CAL
       SAC   COW07
       CAL
       AND   COW04
       ADD   ZERO
       SAC*  COW8
       ISZ   COW06
       JMP   COW01
       JMP*  COW4C
       REM
**************************************************************************
COW04  OCT   0007
COW05  DEC   -4
COW06  DEC   00           COUNTER
COW07  DEC   00           STORES WORD
COW8   EQU   10
**************************************************************************
*
       EJECT
```

```
**********************************************************************
       REM    ERRTN          ERROR ROUTINE
**********************************************************************
       REM    ENTRANCE..     JMS*  ERROR
       REM                   DEC   N (ERROR NUMBER)
**********************************************************************
ERRTN  NOP
       LAC*   ERRTN          STORE ERROR NUMBER
       ADD    ZERO
       SAC    ER006
       JMS*   *+1
       DSA    COW4C
       DSA    ER006+1
       DSA    CC
       LAC    ER007          SET COUNTER
       SAC    ER008
       LAC    ER009          SET ADRESS
       SAC    ER8
ER001  LAC*   ER8            CHARACTER TO AC
       TSF                   PRINT
       JMP    *-1
       TLS
       ISZ    ER008
       JMP    ER001          NEXT CHARACTER
       ISZ    ERRTN
       JMP*   ERRTN
       REM
**********************************************************************
ER005  OCT    215            CARRIAGE RETURN
       BCI    6 ERROR
ER006  DEC    00             ERROR NUMBER
       DEC    00
       DEC    0
       DEC    0
       DEC    0
       DEC    0
ER007  DEC    -13
ER009  DSA    ER005-1
ER008  BSS    1              COUNTER
ER8    EQU    8
**********************************************************************
*
       EJECT
```

```
**********************************************************************
       REM    ENTRANCE..  JMS*  *+1
       REM                DSA   READ
**********************************************************************
       REM    ACTION..    BLANKS LINE+1 TO LINE+72.
       REM                LINE FEED SIGNIFIES A NEW LINE
       REM                CARRIAGE RETURN IS TAKEN AS END OF A LINE
       REM                LINE FEED AND CARRIAGE RETURN ARE NOT STORED.
**********************************************************************
       OGS
READ   NOP
       KSP                IS PAPER TAPE READY
       JMP    *-1         NO
       KRP                YES READ A CHARACTER
       CAS    0212        IS IT A LINE FEED
       JMP    READ+1      NO
       JMP    *+2         YES. LINE FEED
       JMP    READ+1      NO
       LAC    D72M        COUNTER
       SAC    READ9
       REM                READ CARD
       LAC    LINEA       ADRESS OF BUFFER
       SAC    READ8
READ2  KSP                IS PAPER TAPE READY
       JMP    *-1         NO
       KRP                YES READ CHARACTER
       CAS    0215        CARRIAGE RETURN
       JMP    *+2         NO
       JMP    READ3       YES
       SAC*   READ8       NO
       ISZ    READ9
       JMP    READ2       NEXT CHARACTER
READ3  CLA                BLANK REAT OF BUFFER
       SAC*   READ8
       ISZ    READ9
       JMP    READ3
       ISZ    READ
       JMP*   READ
       REM
**********************************************************************
LINEA  DSA    LINE
READ8  EQU    8
D72M   DEC    -72
READ9  BSS    1
LINE   BSS    73
**********************************************************************
*
       EJECT
```

```
**********************************************************************
      REM   COMMONLOCATIONS IN SECTOR ZERO
**********************************************************************
      ORG   16
STAR  BCI   1 *
BLANK BCI   1
ZERO  BCI   1 0
0260  EQU   ZERO
ONECH BCI   1 1
SEVEN BCI   1 7
NINE  BCI   1 9
PLUS  BCI   1 +
MINUS BCI   1 -
A     BCI   1 A
0212  OCT   212          LINE FEED
0215  OCT   215
CR    EQU   0215         CARRIAGE RETURN
ONE   DEC   1
ERROR DSA   ERRTN        ADRESS OF ERROR ROUTINE
CC    BSS   1            CARD COUNT
LC    BSS   1            LOCATION COUNTER
AI    BSS   1            ASSEMBLED INSTRUCTION
WORD  EQU   AI
PASS  BSS   1            0 FOR PASS 1     1 FOR PASS2
P2010 BSS   1
      REM
********
      REM   FIELDS IN INPUT LINE
**********
LF    EQU   LINE+1
OF    EQU   LINE+7
VF    EQU   LINE+13
      REM
      REM
      REM
      REM
      REM
**********
**********
**********
```

```
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
        END    START
```